An Algorithm for Estimating all Matches Between Two Strings

Mikhail J. Atallah*
COAST Laboratory and
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
U.S.A.
mja@cs.purdue.edu

Frédéric Chyzak†
INRIA
Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Frederic.Chyzak@inria.fr

Philippe Dumas
INRIA
Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Philippe.Dumas@inria.fr

Abstract

We give a randomized algorithm for estimating the score vector of matches between a text string of length N and a pattern string of length M; this is the vector obtained when the pattern is slid along the text, and the number of matches is counted for each position. The randomized algorithm takes deterministic time $O((N/M)Conv(M))$ where $Conv(M)$ is the time for performing a convolution of two vectors of size M each. The algorithm finds an unbiased estimator of the scores, whose variance is particularly small for scores that are close to M, i.e., for approximate occurrences of the pattern in the text. No assumptions are made about the probabilistic characteristics of the input, or about the number of different symbols appearing in T or P (i.e., the alphabet size need not be much smaller than M). The solution extends to the weighted case and to higher dimensions.

Index Terms — algorithms, convolution, pattern matching.

*Portions of this work were supported by sponsors of the COAST Laboratory.

†The second and third authors' work was supported in part by the Long Term Research Project Alcom-IT (#20244) of the European Union.

1 Introduction

We address the following problem: let T be a text string and P be a pattern string defined by

$$T = t_0 t_1 \dots t_{N-1} \qquad \text{and} \qquad P = p_0 p_1 \dots p_{M-1}.$$

We want to compute the *score of matches between T and P*, i.e., the vector C whose ith component c_i is the number of matches between the text and the pattern when the first letter of the pattern is positioned in front of the ith letter of the string (see Figure 1). Formally, for $i = 0, \cdots, N - M$,

$$c_i = \sum_{j=0}^{M-1} \delta_{t_{i+j}, p_j}$$

where $\delta_{x,y}$ denotes the Kronecker symbol: $\delta_{x,y}$ is 1 if and only if $x = y$ and is 0 otherwise.

Computing the score vector solves one version of the problem of approximate pattern matching: an exact match corresponds to a score $c = M$; a match with e errors to a score $c = M - e$. Rather than focusing on computing the exact scores, we develop an efficient randomized algorithm to approximate them. This algorithm can be tuned to attain an arbitrary level of accuracy and, besides, the fewer the number e of mismatches, the better is the approximation that the algorithm returns: even if the estimated score can be far from the exact value when the pattern and the text have little match, an almost complete match will be recognized by the algorithm. The algorithm locates these interesting positions with good accuracy, which is the difficult part of the problem.

Approximate pattern matching has many applications, including intrusion detection in a computer system [12], image analysis and data compression [4]. In the former, alphabet symbols correspond to events in a system, and since some events are more important than others (from a security point of view) it follows that the definition of the score needs to be *weighted* by the relative importance of alphabet symbols. This led us to generalize our method and result so that they apply to weighted versions of the problem, i.e., to the problem of computing weighted scores defined by

$$c_i = \sum_{j=0}^{M-1} f(t_{i+j}) g(p_j) \delta_{t_{i+j}, p_j},$$

where f and g are complex-valued functions of the alphabet.

The naive algorithm to compute the exact score vector has a time complexity of $O((N - M + 1)M)$. When the alphabet size is $O(1)$ (hence much smaller than M), the algorithm of Fisher and Paterson [9] uses convolution to solve the problem in $O(N \log M)$ time. However, if the assumption of small alphabet size is dropped, then another approach is needed. This version of the problem (i.e., for possibly large alphabets) was posed by Apostolico and Galil in their book [2], where it is mentioned that a linear time algorithm can be obtained for computing those offsets i at which only a single mismatch prevents the pattern from occurring exactly. (The corresponding entries of C then equal $M - 1$.) The best known deterministic algorithm for computing the vector C is due independently to Abrahamson and Kosaraju [3, 11] and has a time complexity of $O\left(N\sqrt{M \log M}\right)$ in the arithmetic computational model in which the convolution of two M-length vectors can be done in $O(M \log M)$ time. The algorithm of Baeza-Yates and Gonnet [6] solves the problem in $O(NM \log M / \log N)$ time, which is better than $O(N \log M)$ for small M, i.e., if $M \leq \log N$. The algorithm of Baeza-Yates and Perleberg [7] solves the problem in average time $O(NM/\sigma)$ where σ is the size of the alphabet (i.e., the number of distinct symbols that appear in M), which is good for large σ. The interest in the vector C is usually motivated by the need to find all positions in the text at which the pattern *almost occurs*, i.e., the offsets i such that c_i is close to M. An algorithm of

Position					i									
Text	...	b	c	a	a	b	c	a	a	b	b	b	a	c	...
Pattern					a	b	a	b	b	a				
Matches							↑		↑					

Figure 1: The pattern is slid along the text and for each position, we count the number of matches between the pattern and the corresponding slice of the text; this gives the score C.

probabilistic time $O(N \log M)$ for this problem was given in [5]; however, this algorithm depends on some restrictive assumptions on the probabilistic characteristics of the input, namely the Bernoulli model. (An earlier version of [5] erroneously claimed that such an assumption is not needed by that algorithm, whereas in fact it was needed.) A clever $O(N(\log M)^3)$ deterministic time algorithm for estimating all the scores of mismatches (rather than of matches) was given by Karloff [10]; although Karloff's estimator is biased, it guarantees not to overestimate the number of mismatches by more than a constant multiplicative factor, and the author states that his scheme apparently cannot be modified to estimate the number of matches (rather than mismatches) to within a constant multiplicative bound.

In this paper we give a randomized algorithm for computing an unbiased estimator of the number of matches:

- Our algorithm runs in deterministic time $O((N/M)Conv(M))$, where $Conv(M)$ is the time it takes to perform the convolution of two vectors of length M all of whose entries are of the form ω^j, where j is an integer and ω is any primitive σth root of unity (recall that σ is the number of distinct symbols that appear in P). Note that this ω is not related to the roots of unity used in the Fourier transform implementation of convolution (the order of these roots of unity depends on M rather than on σ, which is typically smaller than M). In the rest of the paper, we replace $Conv(M)$ by $M \log M$, which corresponds to the computational model where an arithmetic operation (addition or multiplication) takes constant time. Our experimental implementations use $\omega = e^{2\pi\sqrt{-1}/\sigma}$, in spite of the roundoff error that this introduces in any realistic computer — the experimental results show that the roundoff error causes no apparent loss of validity of the theoretical predictions (i.e., they confirm the theory).

- The algorithm is randomized but its behaviour does not depend on any a priori probabilistic assumption on the input, or the size of the alphabet. The expected value of the estimator is equal to the exact value. More precisely, we compute our estimate of the score vector $\widehat{C}$ in deterministic time $O(kN \log M)$, such that the expected value of its ith component $\widehat{c_i}$ equals c_i. Moreover, the standard deviation is bounded above by $(M - c_i)/\sqrt{k}$. Note that we have a trade-off between time complexity and accuracy: by choosing larger values of k, more accurate estimates are obtained. Also note that the standard deviation is particularly small when c_i is close to M, which is precisely the case of *almost occurrence* that usually interests us.

We summarize our main results is the following theorem.

Theorem 1 *An estimate for the score C between a text string of length N and a pattern string of length M can be computed by a Monte-Carlo algorithm in time $O(N \log M)$. The randomized result has mean C and each entry has a variance bounded by $(M - c_i)^2/k$ where k is the number of iterations in the Monte-Carlo algorithm.*

Although we feel that the algorithm should work best with the FFT step performed by dedicated chips, we have a full implementation including a soft FFT. First experiments confirm the theory, and are described in the last section.

2 Preliminaries and terminology

We first observe that it suffices to obtain an algorithm of time complexity $O(kM \log M)$ for the case $N = 2M$. Indeed, if $N > 2M$, we can use the standard technique [8] of partitioning the text into $O(N/M)$ overlapping chunks of length $2M$ each, and then processing each chunk separately in time $O(kM \log M)$. The overall complexity is then $O(kN \log M)$. Therefore we henceforth assume, without loss of generality, that $N = 2M$.

Let $\mathcal{A}$ be a finite alphabet of cardinality σ. We use the following notation for the integer interval

$$[0, \sigma[= \{0, \cdots, \sigma - 1\}.$$

We henceforth use ω to denote any primitive σth root of unity and Σ to denote the set of all possible mappings from $\mathcal{A}$ to $[0, \sigma[$. Observe that, for a random variable X that is uniformly distributed over $[0, \sigma[$, we have $\mathrm{E}\left(\omega^X\right) = 0$. This fact stems from the nullity of the sum of all the σth roots of unity. As a corollary, if the random variable Φ is uniformly distributed over Σ, then $\mathrm{E}\left(\omega^{\Phi(a)}\right) = 0$ for any $a \in \mathcal{A}$.

3 The algorithm

The key idea of our algorithm is to iteratively compute randomized estimated values for C; the values obtained are those of a random variable whose mean is the score and whose variance is small. We then repeat the calculation to estimate the score with good probability. In the sequel, k denotes a positive integer, the number of repetitions of the algorithm. As will be shown in the next section, the larger k, the smaller the variance of our answers.

The iteration step of the algorithm is based on the following idea: assume that we have two strings of length M; if we renumber the letters at random with numbers from $[0, \sigma[$, we obtain two integer sequences $n_0 \dots n_{M-1}$ and $m_0 \dots m_{M-1}$; in the mean over all renumberings, the Hermitian inner product

$$\sum_{j=0}^{M-1} \omega^{n_j} \overline{\omega^{m_j}} = \sum_{j=0}^{M-1} \omega^{n_j - m_j}$$

counts the number of matches between both strings. We therefore have the following algorithm to compute the score:

Algorithm MATCH
INPUT: a text $T = t_0 \dots t_{2M-1}$ and a pattern $P = p_0 \dots p_{M-1}$ where the t_i's and the p_i's are letters from $\mathcal{A}$;
OUTPUT: an estimate for the score vector C.

1. For $\ell = 1, 2, \cdots, k$:
   (a) select randomly and uniformly a $\Phi^{(\ell)}$ from Σ;
   (b) from the text T, obtain a complex sequence $T^{(\ell)}$ of size $2M$ by replacing every symbol t in T by $\omega^{\Phi^{(\ell)}(t)}$;

(c) from the pattern P, obtain a complex sequence $P^{(\ell)}$ by

i. replacing every symbol p in P by $\omega^{-\Phi^{(\ell)}(p)}$;

ii. padding with M (trailing) zeroes;

(d) compute the vector $C^{(\ell)}$ as the convolution of $T^{(\ell)}$ with the reverse of $P^{(\ell)}$, i.e.,

$$c_i^{(\ell)} = \sum_{j=0}^{M-1} \omega^{\Phi^{(\ell)}(t_{i+j})} \overline{\omega^{\Phi^{(\ell)}(p_j)}} = \sum_{j=0}^{M-1} \omega^{\Phi^{(\ell)}(t_{i+j}) - \Phi^{(\ell)}(p_j)};$$

2. compute the vector $\widehat{C} = \sum_{\ell=1}^{k} C^{(\ell)}/k$ and output it as our estimate of C.

The previous algorithm deserves several remarks.

- It is crucial that $\Phi^{(\ell)}$ be a random *mapping* rather than a random *permutation*. In fact, developing the analysis of the next section with a permutation rather than a mapping would reveal that the corresponding estimate is biased, whereas we need an unbiased estimate of C.
- The computation of the convolution is performed by fast Fourier transform. The time complexity of this classical algorithm is $O(M \log M)$, which makes us achieve a low complexity for the overall algorithm. Besides, this algorithm is now implemented in dedicated chips.
- The fast Fourier transform evaluates polynomials related to its inputs at roots of unity. The latter are not related to ω; their order is not σ but the size of the text $2M$.
- Of course, when implementing the algorithm, one should use the same array for all the $C^{(\ell)}$'s, so as to achieve a space complexity of $O(M)$ rather than $O(kM)$. (The time complexity is left unchanged by this optimization.)

4 Analysis

The analysis in the next section will show that $\mathrm{E}\left(\widehat{C}\right) = C$, and that the standard deviation of $\widehat{c}_i$ is bounded above by $(M - c_i)/\sqrt{k}$. One is usually only interested in the positions i at which c_i is very close to M (i.e., where the pattern almost occurs in the text). A fact of interest is that it is precisely for these i's that the standard deviation is the smallest. Therefore $\widehat{c}_i$ is a particularly good estimator of c_i when c_i is close to M. Here by *close to M* we mean a high percentage of agreement, i.e., $c_i = \lambda M$ where λ is a constant close to 1.

We proceed to estimate the mean and the variance of the $\widehat{c}_i$'s. All the random variables $\widehat{c}_i$ are defined in a similar way; hence we generically consider the random variable

$$\widehat{s} = \frac{1}{k} \sum_{\ell=1}^{k} \sum_{j=0}^{M-1} \omega^{\Phi^{(\ell)}(t_j) - \Phi^{(\ell)}(p_j)},$$

where the t_j's and the p_j's are fixed and the $\Phi^{(\ell)}$'s are independent and uniformly distributed random mappings from $\mathcal{A}$ to $[0, \sigma[$. The number of matches between $t_0 \ldots t_{M-1}$ and $p_0 \ldots p_{M-1}$ is given by

$$c = \sum_{j=0}^{M-1} \delta_{t_j, p_j},$$

where once again $\delta_{x,y}$ denotes the Kronecker symbol.

The random variable $\widehat{s}$ is the mean of k independent identically distributed random variables $s^{(\ell)}$. Hence it suffices to consider the random variable

$$s = \sum_{j=0}^{M-1} \omega^{\Phi(t_j)-\Phi(p_j)},$$

for the mean and variance of $\widehat{s}$ are then given by

$$\mathrm{E}\left(\widehat{s}\right) = \mathrm{E}(s) \qquad \text{and} \qquad \mathrm{Var}\left(\widehat{s}\right) = \frac{\mathrm{Var}(s)}{k}.$$

We start by evaluating the mean of $\widehat{s}$ with the following lemma.

Lemma 1 *The mean of $\widehat{s}$ is the number c of matches between $t_0 \dots t_{M-1}$ and $p_0 \dots p_{M-1}$.*

Proof. The mean of $\widehat{s}$ is given by

$$\mathrm{E}\left(\widehat{s}\right) = \mathrm{E}(s) = \sum_{j=0}^{M-1} \mathrm{E}\left(\omega^{\Phi(t_j)-\Phi(p_j)}\right).$$

Now, observe that the mean inside the sum is zero unless $t_j = p_j$, because $\omega^{\Phi(t_j)-\Phi(p_j)}$ is equally likely to be any of the σth roots of unity and the sum of all the σth roots of unity is zero. More precisely, we have the equality

$$\mathrm{E}\left(\omega^{\Phi(t_j)-\Phi(p_j)}\right) = \delta_{t_j,p_j},$$

from which the result follows. □

Next, we consider the variance of $\widehat{s}$. We only state the result for now, and postpone its proof until the next section (where a more general version is proved).

Lemma 2 *The variance of $\widehat{s}$ is bounded as follows:*

$$\mathrm{Var}\left(\widehat{s}\right) \leq \frac{(M-c)^2}{k}.$$

Theorem 1 now follows from Lemmas 1 and 2.

5 Generalizations

The previous technique extends to two directions: we can consider weighted versions of the problem by using a more general function than the characteristic function of matches; we can consider arrays in place of words, or more generally higher dimensional arrays.

5.1 Weighted case

The method and results we developed apply to weighted versions of the problem, i.e., to the problem of computing weighted scores defined by

$$c_i = \sum_{j=0}^{M-1} f(t_{i+j}) g(p_j) \delta_{t_{i+j},p_j},$$

where f and g are complex-valued functions of the alphabet.

In the algorithm, the encoding of the alphabet using roots of unity has to be changed accordingly: when creating $T^{(\ell)}$ we now replace every symbol t in T by $f(t)\omega^{\Phi^{(\ell)}(t)}$, while when creating $P^{(\ell)}$ we replace every symbol p in P by $g(p)\omega^{-\Phi^{(\ell)}(p)}$.

As a matter of fact, we proceed to perform our analysis in the more general case of weighted scores of the form

$$c_i = \sum_{j=0}^{M-1} h(t_{i+j}, p_j)\delta_{t_{i+j},p_j},$$

where h is a complex-valued function on pairs of letters in $\mathcal{A}^2$; we do this essentially for the purpose of analysis, and it entails no loss of generality since our algorithm deals with is the special case of $h(t_{i+j}, p_j) = f(t_{i+j})g(p_j)$. The randomized vector $\widehat{C}$ we obtain still has the property to be C, while the variance of $\widehat{c_i}$ is now $O((M - c_i)/\sqrt{k})$.

Once again, we generically consider the random variable

$$\widehat{s} = \frac{1}{k}\sum_{\ell=1}^{k}\sum_{j=0}^{M-1} h(t_j, p_j)\omega^{\Phi^{(\ell)}(t_j)-\Phi^{(\ell)}(p_j)},$$

where the t_j's and the p_j's are fixed and the $\Phi^{(\ell)}$'s are independent and uniformly distributed random mappings from $\mathcal{A}$ to $[0, \sigma[$. The weighted score between $t_0 \dots t_{M-1}$ and $p_0 \dots p_{M-1}$ is given by

$$c = \sum_{j=0}^{M-1} h(t_j, p_j)\delta_{t_j,p_j},$$

where once again $\delta_{x,y}$ denotes the Kronecker symbol.

The random variable $\widehat{s}$ is the mean of k independent identically distributed random variables $s^{(\ell)}$. Hence it suffices to consider the random variable

$$s = \sum_{j=0}^{M-1} h(t_j, p_j)\omega^{\Phi(t_j)-\Phi(p_j)},$$

for the mean and variance of $\widehat{s}$ are then given by

$$\mathrm{E}(\widehat{s}) = \mathrm{E}(s) \qquad \text{and} \qquad \mathrm{Var}(\widehat{s}) = \frac{\mathrm{Var}(s)}{k}.$$

The analysis differs from the unweighted case in that the role of $\delta_{x,y}$ in the unweighted case is now played by $h(x, y)\delta_{x,y}$. We start with the mean.

Lemma 3 *The mean of $\widehat{s}$ is the weighted score*

$$c = \sum_{j=0}^{M-1} h(t_j, p_j)\delta_{t_j,p_j}$$

between $t_0 \dots t_{M-1}$ and $p_0 \dots p_{M-1}$.

Proof. The mean of $\widehat{s}$ is given by

$$\mathrm{E}(\widehat{s}) = \mathrm{E}(s) = \sum_{j=0}^{M-1} \mathrm{E}\left(h(t_j, p_j)\omega^{\Phi(t_j)-\Phi(p_j)}\right) = \sum_{j=0}^{M-1} h(t_j, p_j)\mathrm{E}\left(\omega^{\Phi(t_j)-\Phi(p_j)}\right) = c,$$

since $\mathrm{E}\left(\omega^{\Phi(t_j)-\Phi(p_j)}\right) = \delta_{t_j,p_j}$. □

We now turn to the variance, proving Lemma 2 as a particular case.

Lemma 4 *The variance of $\widehat{s}$ is bounded as*

$$\mathrm{Var}\left(\widehat{s}\right) \leq \frac{||h||_\infty (M-c)^2}{k},$$

where $||h||_\infty$ denotes the maximum value of $|h(x,y)|$ over $\mathcal{A}^2$.

Proof. Since $\mathrm{Var}(s) = \mathrm{E}\left(|s|^2\right) - |\mathrm{E}(s)|^2$, we first study the mean of $|s|^2 = s\overline{s}$. It is

$$\mathrm{E}\left(s\overline{s}\right) = \sum_{0 \leq i,j < M} h(t_i,p_i)\overline{h(t_j,p_j)}\mathrm{E}\left(\omega^{\Phi(t_i)-\Phi(p_i)-\Phi(t_j)+\Phi(p_j)}\right).$$

When $\omega^{\Phi(t_i)-\Phi(p_i)-\Phi(t_j)+\Phi(p_j)} = 1$ independently from Φ, the inner mean $\mathrm{E}\left(\omega^{\Phi(t_i)-\Phi(p_i)-\Phi(t_j)+\Phi(p_j)}\right)$ is 1; otherwise, it is 0. By a simpler inclusion-exclusion argument, it follows that

$$\mathrm{E}\left(s\overline{s}\right) = \sum_{0 \leq i,j < M} h(t_i,p_i)\overline{h(t_j,p_j)}\left(\delta_{t_i,p_i}\delta_{t_j,p_j} + \delta_{t_i,t_j}\delta_{p_i,p_j} - \delta_{t_i,t_j}\delta_{p_i,p_j}\delta_{t_i,p_i}\delta_{t_j,p_j}\right).$$

With the first product of Kronecker symbols, one recognizes the expansion of $|\mathrm{E}(s)|^2$, so that

$$\mathrm{Var}(s) = \mathrm{E}\left(|s|^2\right) - |\mathrm{E}(s)|^2 = \sum_{0 \leq i,j < M} h(t_i,p_i)\overline{h(t_j,p_j)}\delta_{t_i,t_j}\delta_{p_i,p_j}\left(1 - \delta_{t_i,p_i}\delta_{t_j,p_j}\right).$$

Let us introduce the real symmetric matrix $\Gamma = [\gamma_{i,j}]$ of size $\sigma \times \sigma$ with (i,j)th entry given by

$$\gamma_{i,j} = \delta_{t_i,t_j}\delta_{p_i,p_j}\left(1 - \delta_{t_i,p_i}\delta_{t_j,p_j}\right),$$

and the vector H with ith entry $h(t_i,p_i)$. We obtain $\mathrm{Var}(s) = \overline{H}^T\Gamma H$, where T denotes the transpose of matrices. Call $\rho(\Gamma)$ the spectral radius of Γ, i.e., the largest modulus of its eigenvalues. Since Γ is positive semidefinite, its eigenvalues are non-negative and $\rho(\Gamma)$ is the largest eigenvalue. We have

$$\mathrm{Var}(s) = \overline{H}^T\Gamma H \leq \rho(\Gamma)\overline{H}^T H.$$

To improve on the latter upper bound and make it more explicit, we need to take the number c of matches into account.

The number $\gamma_{i,j}$ is 0 unless $t_i = t_j \neq p_i = p_j$. It entails that in case of a match $t_i = p_i$, both the ith line and the ith column of Γ are 0. After renumbering the lines and columns in Γ and H, we part the latter as follows:

$$\Gamma = \left[\begin{array}{c|c} 0 & 0 \\ \hline 0 & \Gamma' \end{array}\right] \qquad \text{and} \qquad H = \left[\begin{array}{c} 0 \\ \hline H' \end{array}\right],$$

where $\Gamma' = \left[\gamma'_{i,j}\right]$ is a matrix of size $(M-c)\times(M-c)$ and H' is a vector of size $(M-c)$. It follows that

$$\overline{H}^T\Gamma H = \overline{H'}^T\Gamma' H' \leq \rho(\Gamma')\overline{H'}^T H'.$$

On the other hand, the spectral radius $\rho(\Gamma')$ of Γ' is bounded above by the Schur norm $\mathcal{N}(\Gamma')$ which satisfies:

$$\mathcal{N}(\Gamma')^2 = \sum_{1 \leq i,j \leq M-c} |\gamma'_{i,j}|^2 \leq (M-c)^2.$$

Furthermore, denoting $||h||_\infty = \max_{(x,y)\in\mathcal{A}^2} |h(x,y)|$, we obtain

$$\overline{H'}^T H' \leq ||h||_\infty^2 (M-c).$$

Finally,

$$\mathrm{Var}\,(\widehat{s}) = \frac{\mathrm{Var}(s)}{k} = \frac{\overline{H}^T \Gamma H}{k} = \frac{\overline{H'}^T \Gamma' H'}{k} \leq \frac{||h||_\infty^2 (M-c)^2}{k}.$$

□

Both lemmas above prove the following theorem.

Theorem 2 *For the weighted version of the problem, an estimate $\widehat{C}$ can be computed by a Monte-Carlo algorithm in time $O(kN\log M)$ with mean and variance given by*

$$\mathrm{E}\left(\widehat{C}\right) = C \qquad \text{and} \qquad \mathrm{Var}(\widehat{c_i}) \leq \frac{||h||_\infty^2 (M-c_i)^2}{k}.$$

Note that the variance is once again particularly small when c_i is close to M.

5.2 Higher dimensional arrays

We sketch the extension to two-dimensional arrays in the non-weighted case; similar ideas would extend it to three and higher dimensions, and to mixed weighted higher-dimensional versions as well.

For the sake of simplicity, we assume in the sequel that M and N are the squares of two integers, $M = m^2$ and $N = n^2$. The text T is now a matrix of size $n \times n$, the pattern P is a smaller matrix of size $m \times m$, and the result we seek is an $(n+1-m) \times (n+1-m)$ matrix C where

$$c_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{m-1} \delta_{T_{i+k,j+l}, P_{k,l}},$$

for $0 \leq i, j \leq n-m$. The time to compute our estimate $\widehat{C}$ of C is now $O(kN\log M)$, and we still have $\mathrm{E}\left(\widehat{C}\right) = C$ and $\mathrm{Var}(\widehat{c_{i,j}}) \leq (M - c_{i,j})^2/k$. We next briefly sketch how this is done.

We justify our focus to achieving a time complexity of $O(kM\log M)$ for the case $n = 2m$ by the following standard reduction [8] to this case from the general case $n > 2m$:

- Cover T with N/M overlapping squares $T_{i,j}$ of size $2m \times 2m$ each, where $T_{i,j}$ consists of the square submatrix of T of size $2m \times 2m$ that *begins* (i.e., has its top-left corner) at position (mi, mj) in T. Hence $T_{i,j}$ and $T_{i+1 \bmod n, j+1 \bmod n}$ overlap over a region of T of size $m \times m$, $T_{i,j}$ and $T_{i,j+1 \bmod n}$ overlap over a region of size $2m \times m$, $T_{i,j}$ and $T_{i+1 \bmod n, j}$ overlap over a region of size $m \times 2m$.

- The algorithm for the case $n = 2m$ is then used on each of the N/M pairs $(T_{i,j}, P)$ of text and pattern. It is easy to see that these N/M answers together contain a description of the desired matrix C. The overall time complexity to compute them is $O((N/M)kM\log M) = O(kN\log M)$, as required.

Therefore, we henceforth assume that $n = 2m$.

The extension of the one-dimensional solution to two dimensions works by transforming the two-dimensional problem into a one-dimensional one [8], and in the process introduces "don't care" symbols: that is, if $\mathcal{A}$ is the alphabet for the two-dimensional problem, then the corresponding alphabet for the one-dimensional problem is $\mathcal{A} \cup \{\texttt{\#}\}$ where # is a "don't care" symbol in the sense that, if x or y (or both) equal # then $\delta_{x,y} = 0$ as a convention.

More specifically, from the text matrix T of size $2m \times 2m$, we create the corresponding text vector V by concatenating the rows of T. Thus V has length $4m^2$. From the pattern matrix P of size $m \times m$, we create a pattern vector W of length $2m^2$ by augmenting each of the rows of P by appending to the end of each of them m symbols # and then concatenating the augmented rows. Let K be the score vector with V as text and W as pattern, i.e.,

$$K_i = \sum_{j=0}^{2m^2-1} \delta_{V_{i+j},W_j}$$

for $0 \leq i \leq 2m^2$ and with the understanding that $\delta_{x,y}$ is zero if either x or y equals the special symbol #.

The connection between K and the score matrix C for text T and pattern P is as follows: $c_{i,j}$ equals $K_{2m(i-1)+j}$. Therefore, computing the matrix C reduces to computing the vector K. The computation is not much more complicated by the presence of the new, special # symbol: we simply follow the rules of the algorithm of Section 3 except that, at the place where the algorithm requires to create $\omega^{\Phi(t)}$ (resp. $\omega^{-\Phi(p)}$), we only do so if t (resp. p) is not the # symbol, and we create a 0 instead if t (resp. p) is the # symbol. Hence we use the weighted model introduced in Section 5, with the weight functions

$$f(a) = g(a) = 1$$

for any letter $a \in \mathcal{A}$ except from

$$f(\texttt{\#}) = g(\texttt{\#}) = 0.$$

The results of Section 5 simply lead to the following theorem.

Theorem 3 *For the two-dimensional version of the problem, an estimate $\widehat{C}$ can be computed by a Monte-Carlo algorithm in time $O(kN \log M)$ with mean and variance given by*

$$\mathrm{E}\left(\widehat{C}\right) = C \qquad \text{and} \qquad \mathrm{Var}\left(\widehat{c_{i,j}}\right) \leq \frac{(M - c_{i,j})^2}{k}.$$

6 Implementation and experimentation

We have implemented and tested our algorithm. Our purpose was to experimentally investigate the quality of the approximation, and whether it confirms the theory. We have performed several experiments on several types of data: randomly generated text, sequenced genes, domains in proteins, literature in several natural languages and MIDI encoding of classical music. What is observed is in excellent agreement with the phenomena predicted by the theory. The algorithm behaves well in practice as soon as the pattern is sufficiently large (typically, larger than 32 or 64 bytes), even for a small value of the parameter k that controls repetitions in the algorithm (typically, $k = 3$ suffices). Our studies concern the unweighted model.

A first experiment was performed with a text of 8192 bytes chosen at random according to the uniform distribution over the alphabet of size 256. The first 4096 bytes were picked and a pattern was obtained by modifying at random, so as to keep 4042 matches. For this case, the parameters are $N = 2M = 8192$, $\sigma = 256$, $k = 3$. All the estimated scores were returned together with the corresponding exact scores. Apart from the almost complete match with score 4042, all other positions have a score less than or equal to 59. The result is that the best match is found while the program behaves well on all other shifts.

As another example, we considered the search for approximate occurrences of a clarinet theme of Beethoven's Fifth Symphony[1]. For this experiment, the datas are MIDI code, and the length of the theme is $M = 128$, so that we set $N = 2M = 256$, $\sigma = 128$, $k = 3$. Furthermore, a threshold of $\lambda = 0.5$ is used in order to filter approximate matches (i.e., the program outputs only those matches with $c \geq \lambda M$).

Here are selected parts of the output in a readable form and sorted by decreasing scores:

```
estd = 1.000000; exact= 128/128 = 1.000000; ratio=1.000000
[********************************************************************************************************************************]

estd = 0.753018; exact= 88/128 = 0.687500; ratio=1.095299
[-****-***-****-***-****-****-****-***-****-***-****-****--***--**--***--**--***--**--***--***--***--***--***-****-]

estd = 0.550580; exact= 70/128 = 0.546875; ratio=1.006775
[-------------------*----*-*--*-****-***-****-***-****-***-****-*--*-****-***--****-***--***-*-**--***-****--***-*--**---*]

estd = 0.507331; exact= 69/128 = 0.539062; ratio=0.941137
[*-****-***-****-***-*--*-*--*-****-***-****-***-****-***-*--*--*-***-***--***-***--***-*--*----*-*-------------------]

estd = 0.568381; exact= 65/128 = 0.507812; ratio=1.119273
[-****-***-****-***-****-****-****-***-****-***-****-**----***--**----**--**----*--*---------*-------------------]

estd = 0.526136; exact= 61/128 = 0.476562; ratio=1.104024
[*******************-*--*-*----------------*******************-*--*-----------*-***-***--***-****---***---------------------------]
```

Each block corresponds to a certain position. In each block, the field `estd` gives the estimated score $\widehat{c_i}/M$, the field `exact` gives the exact score c_i/M and the field `ratio` gives the ratio $\widehat{c_i}/c_i$. The characters `*` and `-` represent a match and a mismatch, respectively. Beside the exact occurrence of the theme (first block), we catch several occurrences where the pattern and the text almost match during long sequences (next four blocks), as well as an occurrence where intermediate-sized sequences of exact match are interlaced with sequences of full mismatch (last block). Note the accuracy of the algorithm on this execution: the ratio $\widehat{c_i}/c_i$ varies little around 1. The algorithm is thus quite good at locating interesting events.

Although the purpose of our experiments was to investigate the quality of the approximation rather than the speed of the algorithm, we can make a few comments about speed. For the parameters of the above-mentioned first experiment, the program processed roughly 280 bytes per second and was much slower (by a factor of 4) than that of Baeza-Yates and Gonnet [6]. As mentioned earlier, we used a soft implementation of FFT (which suffers from large constant factors in its time complexity), and our algorithm should work better with the FFT step performed by dedicated chips. Of course for large enough problem sizes the asymptotic time complexity overcomes the effect of large constant factors, but with our current software implementation "large enough" means roughly $N \geq 33,000$ (assuming $M = N/2$, $\sigma = 256$, and $k = 3$). Such large problem sizes actually make it impossible to effectively use dedicated FFT chips to achieve better speeds, because the currently available and planned FFT chips will not "fit" such huge problem sizes. Now, suppose that one has hardware designed for a p-sized FFT problem, and one wants to use it to solve an n-sized FFT problem, where $n > p$. Clever techniques for optimally using the p-sized FFT hardware to solve an n-sized FFT were designed by Aggarwal and Vitter [1]. However, these methods introduce serious practical complications of their own, such as necessitating multiple uses of the dedicated FFT chip, and the elaborate combining of the answers returned by these multiple uses of the FFT chip. This would involve impractical constant factors.

Acknowledgement. It is a pleasure to acknowledge the helpful comments and performance data kindly supplied by the referees.

[1]The authors warmly thank Roberto Sierra (`bert@netcom.com`) who sequenced the whole symphony in MIDI code, together with other musical pieces, and made them freely available on the WEB.

References

[1] A. Aggarwal and J.S. Vitter, "The Input/Output Complexity of Sorting and Related Problems," *Communications of the ACM*, Vol. 31, 1988, pp. 1116–1127.

[2] A. Apostolico and Z. Galil (Eds), *Combinatorial Algorithms on Words,* Springer, 1985.

[3] K. Abrahamson, "Generalized String Matching," *SIAM Journal of Computing*, 16, 1987, pp. 1039–1051.

[4] M.J. Atallah, Y. Génin, and W. Szpankowski, "A Pattern Matching Approach to Image Compression," *Proceedings of the Third IEEE International Conference on Image Processing*, Lausanne, Switzerland, 1996, pp. 349–356.

[5] M.J. Atallah, P. Jacquet, and W. Szpankowski, "A Probabilistic Approach to Pattern Matching with Mismatches," *Random Structures and Algorithms*, 4, 1993, pp. 191–213.

[6] R.A. Baeza-Yates and G.H. Gonnet, "A New Approach to Text Searching," *Communications of the ACM,* 35, 1992, pp. 74–82.

[7] R.A. Baeza-Yates and C.H. Perleberg, "Fast and Practical Approximate Pattern Matching," *Information Processing Letters,* 59, 1996, pp. 21–27.

[8] M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, 1994.

[9] M.J. Fischer and M.S. Paterson, "String Matching and Other Products," *Complexity of Computation, SIAM-AMS Proceedings,* 7, 1974, pp. 113–125.

[10] H. Karloff, "Fast Algorithms for Approximately Counting Mismatches," *Information Processing Letters,* 48, 1993, pp. 53–60.

[11] S.R. Kosaraju, "Efficient String Matching," manuscript, Johns Hopkins University, 1987.

[12] S. Kumar and E.H. Spafford, "A Pattern-Matching Model for Intrusion Detection," *Proceedings of the National Computer Security Conference*, 1994, pp. 11–21.

[13] D.E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms,* Addison-Wesley, 2nd ed., 1981, pp. 290–294.